DEPARTMENT OF DEFENCE
DEFENCE SCIENCE & TECHNOLOGY ORGANISATION **DSTO**

A Federated Geospatial and Imagery Exploitation Service (GIXS) Model

Derek Weber and Heath James

DSTO-TR-1013

A Federated Geospatial and Imagery Exploitation Service (GIXS) Model

Derek Weber

Surveillance Systems Division
Electronics and Surveillance Research Laboratory

Heath James

Distributed High Performance Computing Group
University of Adelaide

DSTO-TR-1013

ABSTRACT

In order for the Geospatial and Imagery Exploitation Service (GIXS) architecture to take advantage of distributed processing of image exploitation tasks, it needs to be adapted to suit a federated environment. This document reports on work in progress by the Image Analysis and Exploitation Group in conjunction with the Distributed and High Performance Computing Group of The University of Adelaide to develop a federated GIXS architecture along with a proof-of-concept implementation.
A federated GIXS model is described, along with a use case scenario including an event-flow diagram. Also described are the changes necessary to adapt the current GIXS standard to our federated model. The report concludes with some future directions for our research.

RELEASE LIMITATION

Approved for public release

DEPARTMENT OF DEFENCE
DEFENCE SCIENCE & TECHNOLOGY ORGANISATION DSTO

AQ F01-09-1692

Published by

DSTO Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury South Australia 5108 Australia

Telephone: (08) 8259 5555
Fax: (08) 8259 6567
© Commonwealth of Australia 2000
AR-011-533
August 2000

A Federated Geospatial and Imagery Exploitation Service (GIXS) Model

Executive Summary

This paper details work in progress on developing and demonstrating a federated image exploitation services model based on the Geospatial and Imagery Exploitation Service (GIXS)[1] architecture. This work is being carried out as part of a collaborative research effort between the Image Analysis and Exploitation Group of the Australian Defence Science and Technology Organisation and the Distributed and High Performance Computing (D&HPC) Group of the University of Adelaide. A conceptual model has been developed and work has commenced on a prototype implementation. Although there are issues still to be worked through, the main concepts are in place. This document describes those concepts, provides an example of the event flow of a given scenario, lists the differences and changes to the current GIXS, and finally itemises some areas for further investigation.

The GIXS architecture is a software model for developing digital image exploitation systems. The GIXS specification is limited, however, in that it implicitly requires that systems using its architecture not only must run on a single host, but cannot divide processing amongst multiple hosts.

An alternative to a single-host architecture is a federated architecture in which multiple hosts are able to share their processing. Federated and distributed software systems have obvious advantages over single-host systems, such as failure recovery capabilities, load balancing, and the potential to parallelise the processing of multi-stage tasks. Although these advantages come at a price (system complexity), distributed systems can make better and more efficient use of resources that already exist, rather than requiring the purchase of more powerful and expensive computers.

The proposed federated GIXS model retains the main subsystems of the current GIXS, however changes have been made to the visibility, some behaviour and responsibilities of the subsystems, along with a few minor interface changes. The proposed model remains interoperable with the United States Imagery and Geospatial Information Service (USIGS)[2], of which the GIXS is itself a part.

Future work includes the development of a working implementation of our federated model, further development of interfaces and communication protocols, the investigation of the persistence of image processing chains, and examination of the issues involved with the dynamic manipulation of distributed image processing chains during execution.

---

[1] *Geospatial and Imagery Exploitation Service (GIXS) Specification*, Version 1.0, National Imagery and Mapping Agency (NIMA), United States Imagery and Geospatial Information System (USIGS), document number S1010420-A, 22 June 1999

[2] *USIGS Architecture Products Home Page*, National Imagery and Mapping Agency (NIMA), 9 February 2000. Available via the WWW as http://www.nima.mil/sandi/arch/.

Authors

Derek Weber
Surveillance Systems Division

Derek Weber completed a Bachelor of Science and Honours in Computer Science at Flinders University in 1998. Since then he has worked as a Professional Officer in the Systems Surveillance Division investigating the various Java-based image processing packages and their applicability to defence applications. Derek's research interests include distributed and object oriented software systems, software architectures and patterns, agent technology, and mobile code technologies.

Heath James
University of Adelaide

Heath James is a Research Fellow in the Distributed and High Performance Computing Group in the Computer Science Department at the University of Adelaide. Heath completed his PhD in July 1999 and is a member of the IEEE, IEEE Computer Society and ACM. Heath's research interests are in scheduling, wide-area distributed computing and metacomputing middleware for scientific applications.

Contents

List of Figures

1. Introduction

This document presents a federated model of image exploitation services adapted from the Geospatial and Imagery Exploitation Service (GIXS) architecture[1].The GIXS architecture was developed by the National Imagery and Mapping Agency (NIMA) of the United States of America as a component of the United States Imagery and Geospatial Information Service (USIGS)[2]. The GIXS specification details a general architecture for building imagery and geospatial information exploitation systems, including standardised interfaces, data conditions and error types. The current version (1.0) of the specification implies that all of the system's image processing is to occur on a single computer; if this system is to be made available to multiple users concurrently the computer must be a powerful and expensive high-end machine.

In contrast to a single host system, a distributed system makes use of many computers networked together typically controlled by a single 'head' machine. A task submitted to the 'head' machine of the network may be farmed out to and processed by other machines on the network, but the 'head' machine is responsible for the production of the result. A federated system builds on this concept because each machine on the network can act as a 'head' for any job submitted to the system. Tasks can be submitted to any host on the network, and can be divided and farmed out from there, rather than having a single point-of-failure of the distributed system (i.e. if the 'head' host goes down, the system is unusable). In a sense a federated system may be thought to be a symmetrically distributed system. A federated system offers many advantages over a single-host system including a potentially better cost/performance ratio and better failure recovery facilities. We introduce the concept of federation to the GIXS, adapting it to a federated system. Please note that this is a work in progress and details are subject to change as research results become available.

The model described in this document makes use of distributed processing techniques through the federation of distributed image exploitation services. A general image exploitation task may consist of a number of operations chained together requiring information from a number of sources and perhaps generating a number of results. This chain can be represented as a *directed acyclic graph* (DAG) of operation nodes. By using distributed processing techniques, different parts of the DAG can be executed on different computers on a network – specific operations could be carried out on dedicated hosts that are particularly well-equipped for them (e.g. a statistics algorithm may only be available on one machine on the network, or a single high-end computer running UNIX may be very good at signal processing operations) if those machines are available at the time of starting execution.

Federated environments offering distributed processing of DAGs of image exploitation operations provide a number of advantages over a single-computer system. These include:

1. *failure recovery* – if one computer on the network fails, then not all work on a particular DAG has been lost;

2. *load distribution* – the processing load of a DAG can be divided equally amongst the computers available on a network, and, depending on the structure of the DAG, it may be possible to parallelise some of the DAG's processing;
3. *network traffic minimisation* – platform independent code (e.g. Java bytecode) can be transferred to a remote host to process large volumes of data (e.g. imagery) to reduce network traffic (instead of the data being moved to a stationary program);
4. *specialisation* – specific computers can be dedicated to particular operations to take advantage of the machine's individual capabilities;
5. *new equipment cost minimisation* – minimise further equipment purchases by making better and more efficient use of current capabilities; and
6. *symmetry* – a federated architecture offers a peer-to-peer symmetry, so that any host may act as the controller for a particular DAG's processing – this avoids the problem of a single point of failure for all DAGs.

We believe that these advantages outweigh those of a single-host system, which may have lower maintenance costs and better individual performance in some circumstances.

1.1 Federation and the USIGS Architecture

The concept of federated systems is already present in the USIGS architecture in the Image Management and Dissemination (IMAD) system[3]. Federation is achieved by making a multitude of image product libraries (IPLs) appear as though they were one library through the use of the IMAD Query Manager. When a query for an image or images is made, it sends the query to the IPLs it knows about and manages the transfer of the results to the requesting machine. If an image is replicated on several IPLs, the Query Manager determines which is the best candidate based on the requesting client's requirements and bandwidth availability. It is straightforward to calculate how complete a request is by checking how much data has arrived at the client's machine; estimates for job times are also readily calculated using bandwidth availability and image size and bit-depth values.

Federating image exploitation through distributing exploitation DAGs over a network of servers presents a variety of problems. Image dissemination involves the transfer of one or more image products to the client's machine. Image exploitation involves the processing of perhaps many images where the various image sources, the various processing services called upon, and the client are potentially dispersed across a wide area network. Updates on progress are significantly more complicated to obtain. It is not clear first of all where the processing is being carried out, secondly whether parts of it are in parallel. Thirdly, since the complexity of image processing algorithms varies all that one can tell about an operation is that the processing is being done. Offering accurate estimates of running time may be especially difficult to do in the situation where the images are of variable size or the computers are not dedicated to image processing tasks.

1.2 Overview of Document

The next section describes the federated model that has been developed so far. Section 3 leads the reader through a distributed processing example as a flow of events. As the example has a straightforward execution path, it is known as the Basic Path scenario. The adaptations and modifications required to alter the current GIXS architecture to our federated model are listed in Section 4. Finally, Section 5 summarises the work to date and indicates where our future research efforts will be directed.

2. A Federated GIXS Model

In developing a federated GIXS model, we have endeavoured to retain as much of the current GIXS architecture as possible. All of the major subsystems of the GIXS are present, however some of the paths of visibility (i.e. which subsystems can see which others and through which interfaces) and subsystem interfaces have been modified. The changes that have been made to date assign the subsystems more precisely defined responsibilities and behaviour. The major areas of change are in the Exploitation Workflow Manager subsystem, the operation of the Exploitation Framework (FW) service, the visibility of the Image Exploitation Services (IESs), and the assumption of the availability of a service broker. The module naming has been altered in accordance with proposed USIGS developments. This section describes the federated GIXS model we have developed. The first subsection is an overview and the following subsections deal with each subsystem individually.

2.1 Overview

Our suggested model is shown in Figure 1. The solid arrows indicate the visibility of subsystems to each other. If a subsystem points to another, it can 'see' its interface. The dashed arrows indicate communication paths over the network via remote method invocations (carried out over middleware such as CORBA[4] or Java's RMI[5]); the direction of the arrows indicate the direction of the method invocations. The Interface Description Language (IDL) boxes indicate which module of IDL is used as the interface to the subsystem. They are defined as described in the GIXS specification, but modifications have been made to some of the modules and interfaces and the Exploitation Query Manager Service (XQM) module is entirely new. The Exploitation Workflow Manager has been replaced by the XQM which comprises the Exploitation Query Manager (XQMgr) itself (the interface to the outside world) and multiple Exploitation Workflow Managers (WFMgrs), one for each exploitation task DAG. The XQMgr represents the Client's (or application's) view of the Exploitation Query Manager Service, and thus of the GIXS system as a whole. The Client may be an application in its own right, or it may be an Exploitation Workflow Manager from a different host (submitting part of a DAG that it cannot process).

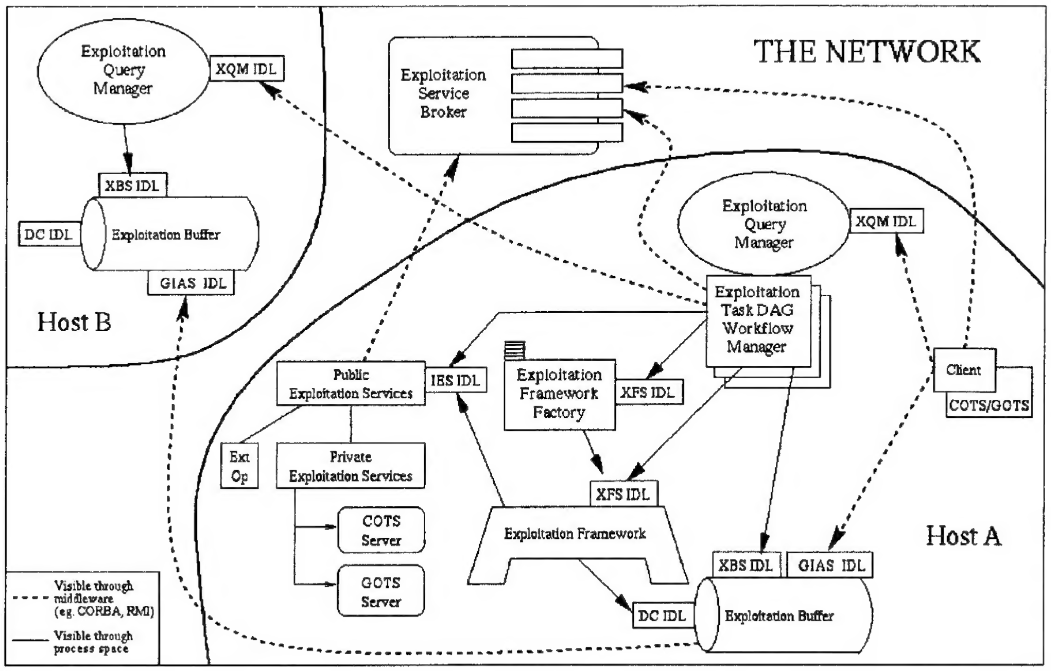
THE NETWORK
Exploitation Query Manager
XQM IDL
XBS IDL
DC IDL
Exploitation Buffer
GIAS IDL
Host B
Exploitation Service Broker
Exploitation Query Manager
XQM IDL
Exploitation Task DAG Workflow Manager
Public Exploitation Services
IES IDL
Exploitation Framework Factory
XFS IDL
Client
COTS/GOTS
Ext Op
Private Exploitation Services
COTS Server
GOTS Server
XFS IDL
Exploitation Framework
Host A
XBS IDL
GIAS IDL
DC IDL
Exploitation Buffer
Visible through middleware (eg. CORBA, RMI)
Visible through process space

Figure 1. The federated GIXS model showing the major subsystems of the architecture and how they are visible to each other

The notion of federation is achieved by the fact that all XQMgrs operate on a client-server basis. If a WFMgr on one host is unable to carry out all processing in an exploitation task DAG, it sends the appropriate sub-DAG to another XQMgr. Knowledge of what services are available on which hosts is obtained through the use of a service broker. DAGs are distributed on the basis of the knowledge in the service broker. Result data is accessed across the network via federated GIAS-based mechanisms such as the IMAD system.

2.2 The Exploitation Query Manager

Each host on the network has a XQMgr running on it that controls the execution of exploitation tasks on that host. Specifically, the XQMgr organises the execution of DAGs submitted to it by Clients, whether the Clients are WFMgrs from other hosts or applications running on the local host. The XQMgr instantiates a WFMgr on a separate thread to manage each DAG, so that the XQMgr can continue receiving and servicing requests as the tasks are carried out. When a WFMgr receives a DAG for processing from its XQMgr it must decide if it can carry out all the operations specified in the DAG, or whether it needs to divide it into sub-DAGs and submit the sub-DAGs that it cannot process to other XQMgrs. A service broker is available for interrogation to determine which hosts can provide the required services, and what their addresses are. A WFMgr manipulates the results of DAGs or sub-DAGs via UID::Product references, which can be used to retrieve actual result data from Geospatial and Imagery Access Service[6] (GIAS)-based systems.

The XQM module currently contains interfaces, data structures, and error conditions for a Client to communicate with a XQMgr. Current interaction is limited to the Client submitting a DAG of image exploitation operations and also the ability to send a callback object reference for the XQMgr to activate once the results of the DAG are available. Future work will focus on (among other areas) interactive management of DAG execution (being able to query the progress of a DAG, pause a DAG, end a DAG in mid-execution, or restart a DAG, for example). The IDL for the XQM module is shown in Figure 2.

```
#include "uco.idl"
#include "uid.idl"

module XQM {

  struct ExceptionInfo { string details; };
  exception DAGIncompletable { ExceptionInfo info; };

  typedef UCO::NameValueList ParameterBlock;
  typedef UCO::NameValueList RenderingHints;
  typedef string DAGRef;

  struct JobInfo {
    string user;
    string group;
    string job_id;
    string priority;
    string permissions;
    RenderingHints global_hints;
  };

  interface XQMCallback {
    notify(in UID::Product prod);
  };

  interface XQManager {
    DAGRef submit_dag(in UCO::DAG expl_dag, in XQMCallback cb)
      throws DAGIncompletable;
  };

}; // end XQM module
```

Figure 2. The XQM module, containing interfaces, data structures, and exceptions for Client communication with an Exploitation Query Manager

2.2.1 Exploitation Workflow Managers

A separate WFMgr thread is created for each exploitation task submitted to the XQMgr by a Client. The WFMgr determines if all processing required can be carried out on the current host, creates sub-DAGs if necessary, and farms out those sub-DAGs to other XQMgrs for processing. A flag may be set to indicate to the other XQMgrs that they should not attempt to optimise the sub-DAGs they get to avoid over-optimisation and continual hand-offs that may lead endless loops. While the Workflow Manager does not have an IDL interface, this is likely to change once dynamic manipulation of DAGs is considered. Currently the XQMgr only instantiates WFMgrs passing them a DAG as a parameter.

It does not have an IDL interface as such, as nothing needs to interact with it other than the XQMgr, and then it is currently only to instantiate it. It may be necessary to introduce an interface to it to provide the ability to dynamically monitor and alter the execution of DAGs.

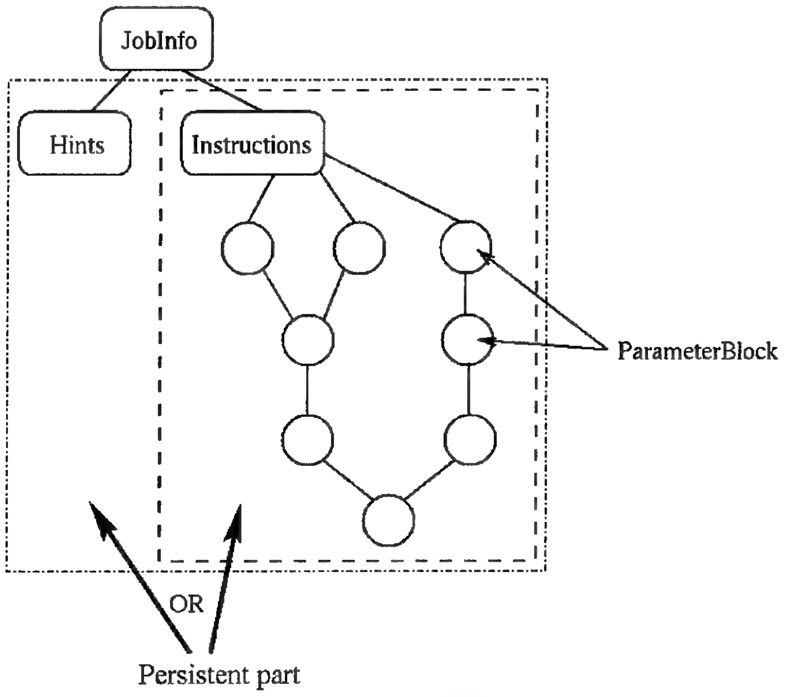

Figure 3. A UCO::DAG structure containing not only the instructions for the image exploitation chain, but also header information including job details and global hints

2.3 DAG Structure

Since the federated GIXS model is intended to be a multi-user system, it is necessary to introduce the notion of ownership to the DAG structure. Currently the nodes in a UCO::DAG structure only contain enough attributes to store the name of an image exploitation operation, a name-value list of arguments to the operation, and an identification number. More attributes are needed on the nodes and in the DAG itself to provide the functionality required. Such attributes include job information (as shown in Figure 2) that can be attached to all sub-DAGs for identification in distributed processing, global hints for all the operations in a DAG, specific hints for each particular operation, and an attribute to store the UID::Product reference to where the results of the DAG should be stored. In addition, the DAG needs to be modifiable, as some information may not be available when the DAG is constructed (e.g. a reference to Buffer space reserved for result data). There also needs to be a mechanism to indicate which intermediate results the Client should be informed of, should they be required. It is quite possible that a user might want to see preliminary processing results before committing to the execution of a long-running exploitation task. The conceptual structure of such a DAG is shown in Figure 3.

In order that an image analyst can save an exploitation task for reuse or to pass on to fellow analysts, the DAG structure must be able to be made persistent. A language created with the Extensible Markup Language (XML) may be an excellent vehicle for this functionality and this is an avenue that we will investigate in the near future.

A draft of the IDL module being used to model the DAG structure and semantics has been developed and included in Appendix B. The DAG is known as an XDAG, and future versions of our federated GIXS architecture may replace the XFS::ExploitationPacket with the XDAG.

2.4 The Exploitation Buffer

To hide the distinction between data stored in the Exploitation Buffer (Buffer) and data stored in GIAS-compliant Image Product Libraries (IPLs), the Buffer has been assigned the extra responsibility of publishing a GIAS interface. In this way, when the Buffer is commanded to load data referred to by a UID::Product reference, it does not need to know whether the data is being retrieved from a remote Buffer or an IPL. In the same way, a Client may retrieve a result from a Buffer or an IPL depending on where the UID::Product reference points.

Furthermore, to allow the creation and manipulation of non-image data in the Buffer, the Data Container (DC) module has been modified to include a NonImageData interface. The IDL for this interface is shown in Figure 4.

```
module DC {

  ...

  interface NonImageData {
    any value;
    string name;
    string type;
    string description;

    NameValueList other_attributes; // eg. Recipe, timestamp
  };

};
```

Figure 4. An IDL interface for non-image data results of image exploitation operations

The XBS::BufferMgr interface has also been modified to include a method for reserving space in the Buffer for result data. The method returns a UID::Product reference to the space, which is passed to a Framework object (Framework). A Framework object, as discussed in section 2.6, carries out the processing of a DAG, converting it to calls to image exploitation services. The Framework uses this reference to store the results of its executions. If the DAG has multiple results, then a hashtable or a name-value map of the results is stored in the space reserved. It is assumed that the Client carrying out the task will know the form of the results. When the Framework has finished executing the DAG, it informs the WFMgr, which returns the result UID::Product reference back to the Client. The signature of the method is:

```
UID::Product reserve_space(in long size_in_bytes);
```

Having a Buffer which acts as a long-term secondary storage (an IPL) as well as acting as short-term storage in working memory does introduce issues about giving disparate responsibilities to one subsystem. A way around this may be to use the Buffer as short-term storage only and to move a product to a colocated IPL after a designated purge time (whether the IPL be colocated on the Buffer's immediate host or just on the LAN). The main reason for the inclusion of a GIAS interface to the Buffer was the need to make the processing results available to the Client. The GIAS interface still does not provide sufficient mechanisms for manipulating non-image data. This is an area for further investigation.

2.5 The Image Exploitation Service

All image exploitation services are accessed through the interfaces in the Image Exploitation Service (IES) module. Specifically this means that any service (for example, rotation and image-to-map overlay operations, or live data-feeds accessed via Java Jini) available to image exploitation tasks must be registered with an IES::OperationFactory. The IES provides image exploitation operations or services, each of which may be *public* (or standard), *private*, or *custom-built* (extended operations). The Image Exploitation Services component[7] of the OpenGIS Abstract Specification[8] may provide a good basis for the *public* services that each host should offer, although the current GIXS specification indicates that the Java Advanced Imaging[9] libraries should be the standard services. In addition to these services is a number of *private* services that can only be accessed on that particular host. For example a particular host may have a RemoteView™[10] server running on it, or it may have capabilities to perform a particular type of processing better than other computers. The IES module also offers the ability to offer custom-built image processing operations. Irrespective of how many of these different types of services and operations are available on each host, their interfaces are all published via the IES::OperationFactory interface. Investigations will take place as to how non-standard (i.e. private and custom-built) operations can register their descriptions with the IES::OperationFactory (although it may be better to leave this to the implementor's discretion). The IES::OperationFactory is also responsible for informing the local service broker of any services that register with it.

The Client discovers what services are available and from where through the use of the local service broker. Although the implementation of the service broker is outside the scope of this project, it is necessary to decide on what information and metadata are to be stored in the service broker. Initial ideas point towards using another XML-based language that can be used to hierarchically categorise services and their features. For example, a rotation service may be classed as an image processing service, which uses a algorithm based on Fourier transforms, has complexity order 3 but has high accuracy, and has an execution time of one minute per 100 x 100 pixel image tile. These details must be query-able, and an XML-based language would be able to provide this given appropriate categories.

The technology required for our service broker could be provided by a technology such as the OpenGIS Catalogue Service[11] or a well-known trading service.

2.6 The Exploitation Framework

A Framework is created on a per-DAG or -use basis via the XFS::FrameworkFactory interface (the XFS::FrameworkMgr renamed). It is a transitory entity whereas the XQMgr, the FrameworkFactory, the Buffer, and the IES::OperationFactory run constantly on a host as server daemons. When the Framework receives a DAG it translates the UCO::DAG representation into a chain of IES::RenderedOps using the structure of the DAG and its content. It also extracts the UID::Product reference to the space reserved for the DAG's results and uses the space when the results are ready. It is the only component of the model that has access to the IES module. Essentially, it is a smart data-pipe, which filters data according to instructions it is given. The sink of the pipe is the space reserved in the Buffer for the results.

The algorithms used by the Framework will be tailored to most efficiently manipulate the DAG structure. The algorithm will need to handle informing the Client of the availability of intermediate exploitation results, and also handle multiple exploitation results (possibly through the use of a hashtable of UCO::Product query keys).

2.7 Federated GIXS and IMAD

The diagram in Figure 5 shows an alternative view of the architecture and how it might interoperate with the IMAD system. It depicts two GIXS systems on different hosts being visible to two different GIXS/GIAS client applications. The IMAD service is also visible to the applications as well as the GIXS systems. All systems use a CORBA backbone as the middleware component used for inter-system communication. Also visible on the system are IPLs, caches of data, and repositories of other data, and the Exploitation Service Broker. Note that this broker may double as a repository for information other than exploitation services. It may advertise data available in IPLs, for example.

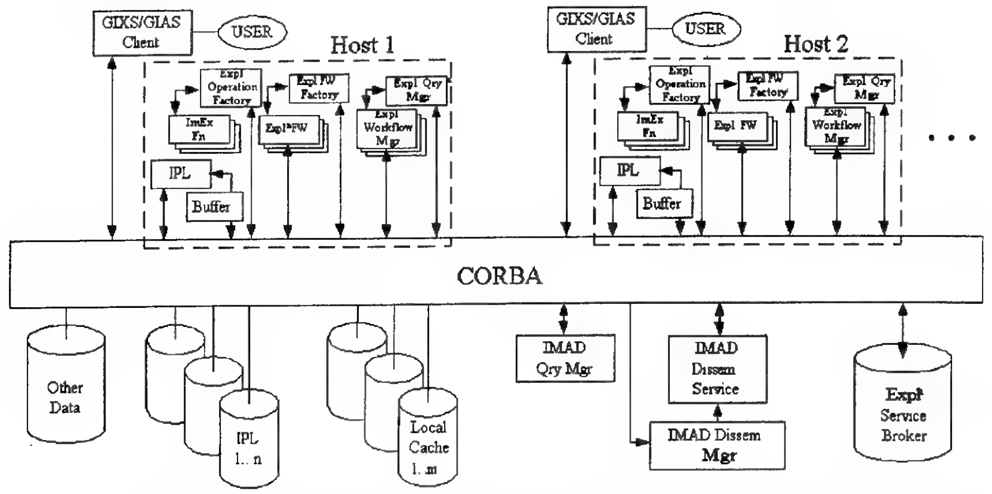

Figure 5. A network-level diagram showing how the federated GIXS system would interact with the IMAD system, and other services.

3. Basic Path Scenario

This section describes the flow of events that occurs when a DAG is submitted to the XQMgr on Host A, as shown in Figure 7. Let us take as an example the DAG shown in Figure 6, consisting of a simple chain with two sources and one result. Assume that Host B has a *composite* operation available that is not available on Host A, but that Host A has a *crop* operation it can use. When the XQMgr on Host A receives the DAG it creates a WFMgr dedicated to managing the task, giving the WFMgr the exploitation DAG and the XQMCallback object submitted by the Client. This allows the XQMgr to continue to receive service requests. The WFMgr examines the DAG and determines that it cannot carry out the first part of the DAG (the *composite* operation). It queries the local service broker to see whether any other hosts offer this service. The service broker informs the WFMgr that Host B can. The WFMgr then separates the DAG into sub-DAGs 1 and 2. It submits sub-DAG 1 to the XQMgr on Host B and waits for it to be executed. When the XQMgr on Host B indicates that it has finished executing, the WFMgr on Host A tells the local Buffer to make the remote result available (via a call to `request_tileable()`) and then starts executing sub-DAG 2 by asking the FrameworkFactory for a Framework and invoking its `start_exploitation()` method. When the result is available, the WFMgr on Host A informs the Client, which retrieves the result from the local Buffer via its GIAS interface. Below is a more detailed flow of events for this scenario.

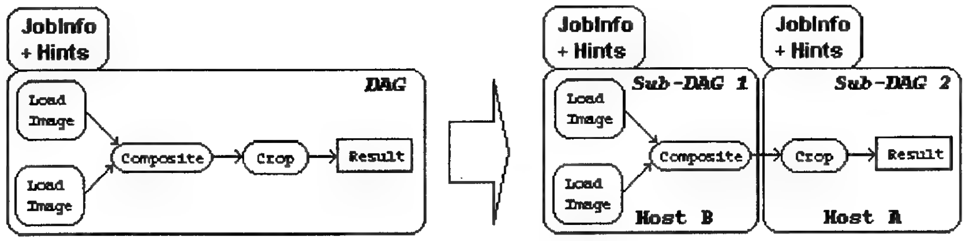

Figure 6. An example DAG of image exploitation operations, and how it is divided into sub-DAGs to be executed on Host A and Host B

Note that the "Load Image" nodes in the above diagram symbolise operations only – the images and their associated metadata may reside on a different host again, even Host A.

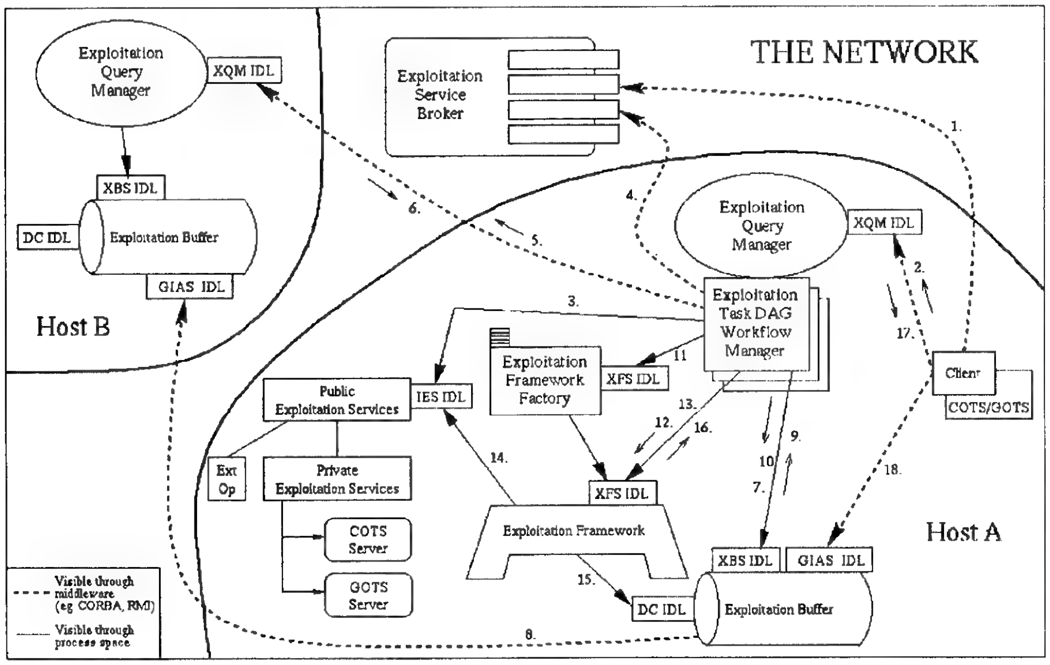
THE NETWORK
Exploitation Query Manager
XQM IDL
XBS IDL
DC IDL
Exploitation Buffer
GIAS IDL
Host B
Exploitation Service Broker
Exploitation Query Manager
XQM IDL
Exploitation Task DAG Workflow Manager
Client
COTS/GOTS
Public Exploitation Services
IES IDL
Exploitation Framework Factory
XFS IDL
Ext Op
Private Exploitation Services
COTS Server
GOTS Server
XFS IDL
Exploitation Framework
XBS IDL
GIAS IDL
DC IDL
Exploitation Buffer
Host A
Visible through middleware (eg CORBA, RMI)
Visible through process space
1.
2.
3.
4.
5.
6.
7.
8.
9.
10
11
12.
13.
14.
15.
16.
17.
18.

Figure 7. A numbered flow of events initiated by a Client submitting a DAG of exploitation tasks to its local XQMgr

The following list identifies the events that occur in according to the numbered arrows:

1. The Client queries the service broker to determine what services are available.
2. The Client then constructs a UCO::DAG using the service descriptions obtained from the service broker. The DAG encapsulates the information shown in Figure 6 as well as job information and global hints. It also constructs an XQMCallback object and submits a reference to it along with the DAG to its local XQMgr.

3. The submitted DAG and callback are passed to a dedicated WFMgr, which manages the execution of the DAG. The WFMgr queries the IES::OperationFactory to determine what services are available locally.
4. It examines the DAG to see if the local system has the capabilities to carry out all the operations in the DAG. When it finds that it does not (it knows it cannot carry out the composition operation) it queries the local service broker to see which hosts can provide the service.
5. When it knows that Host B can provide the service it divides the DAG into sub-DAGs 1 and 2, and attaches the head node of the original DAG to each sub-DAG. The WFMgr then passes sub-DAG 1 to the XQMgr on Host B (using the address returned to it from the service broker) and then waits until it is notified that execution is complete on Host B.
6. When results of the processing of sub-DAG 1 are available, the WFMgr assigned to the task on Host B notifies the WFMgr on Host A that the result is ready and available. It does this via a callback mechanism, passing a UID::Product reference to the result as a parameter to the callback.
7. The WFMgr then commands the Buffer to make the remote result available, passing the UID::Product reference as a parameter to the method `XBS::BufferMgr.request_tilable()`. The task thread also registers an XBS::XBCallback with the Buffer. The callback will be notified when the result is ready for use.
8. The Buffer then prepares the remote result by retrieving the data or opening a data-pipe to the Buffer or GIAS-compliant system holding the result.
9. Once the data is ready the Buffer activates a callback to wake up WFMgr. As a parameter to the callback method (i.e. `notify()`) a UID::Product reference to the local copy of the data is sent to the WFMgr.
10. The WFMgr asks the Buffer to reserve a space for the result of the processing of sub-DAG 2 and receives a UID::Product reference as a return value.
11. The WFMgr annotates sub-DAG 2, adding the reference* to the local copy of the result of sub-DAG 1 as a source of sub-DAG 2, and also adding the reference to reserved space in the local Buffer for the result of the execution. The WFMgr then asks the XFS::FrameworkFactory for a Framework object using the job information included in the original DAG or alternatively extracted from the sub-DAG. The FrameworkFactory returns a reference to a Framework extracted from a pool of them or from a newly instantiated object (the specification of the current XFS::FrameworkMgr seems to require the use of a resource pooling mechanism).
12. The WFMgr registers an EFCallback object with the Framwork object.
13. Execution of sub-DAG 2 is initiated by the WFMgr by calling the Framework's `start_exploitation()` method.
14. The Framework extracts the information for the sub-DAG in order to translate each node to a IES operation. The node contains the name of the operation, along with a UCO::NameValueList of the arguments required by the operation, and also an optional list of hints. It manipulates the data in the form of DC::RenderedImage and

* What format the references passed between the Buffer, the WFMgr, and Frameworks should be is a matter for further investigation. Some of the options available for the reference are UID::Product and UCO::FileLocation, although these may be inadequate.

DC::NonImageData objects. Sub-DAG 2 consists only of a single crop operation. The Framework carries out this operation.

15. The Framework then places the result into the space allocated in the Buffer. This may need to be done via the GIAS interface if required (e.g. we decide that the Buffer should not provide a GIAS interface, and its IPL functions should be provided by a dedicated IPL), but for the moment we do this via the DC module.
16. The Framework then notifies the WFMgr via the EFCallback that the thread had registered when submitting the sub-DAG. This is the end of the life of the Framework object.
17. The WFMgr now notifies the Client via the XQMCallback, passing the UID::Product reference to the local result in the Buffer as a parameter.
18. The Client accesses the final result in the Buffer through its GIAS interface using the UID::Product reference.

If the final result had been calculated on Host B instead, the WFMgr would have passed to the Client the UID::Product reference to the remote result instead of a local reference. To the Client, they would be identical apart from the latency, which could be overcome by having the WFMgr make the result available locally or via an appropriate image dissemination technology. For example, IMAD makes use of multiple resolutions of the same image to get the data to the Client in the most efficient manner. It is also entirely possible that all operations can be carried out on the local host, or equally that all parts of the DAG must be farmed out. This second case may occur if the current host is already too loaded even if it does have the capabilities required. A mechanism needs to be developed that allows hosts to refuse DAGs gracefully rather than continually creating threads to manage them and subsequently sending the DAGs to other hosts to execute (which generates a livelock situation and a memory leak).

4. Changes Required to the Current GIXS

The federated GIXS model presented has been adapted from the current GIXS model. This section explicitly lists the modifications required to move from the current GIXS to the federated model.

The proposed changes are as follows:

- *Overall:*
  - The Exploitation Workflow Manager has been replaced by the Exploitation Query Manager and multiple, single task-dedicated Exploitation Workflow Managers, whose roles and responsibilities have been clearly defined;
  - the paths of visibility between the subsystems must be altered such that only the FW can view the IES, and the Client can only see the XQMgr and the Buffer;
  - the assumption of the availability of a service broker;
- *Exploitation Query Manager:*
  - the introduction of the XQM module and the description of the XQMgr's and the WFMgrs' behaviours and responsibilities;
- *Exlploitation Buffer Service:*

  - the altering of the XBS::BufferMgr to include a method for reserving space in the Buffer for results;
  - the addition of the responsibility of publishing a GIAS interface
  - the addition of the responsibility of 'loading' or making data available through the `XBS::BufferMgr.request_[un]tileable()` methods;
- *Data Containers:*
  - the introduction of the NonImageData interface and the addition of the `create_non_image_data()` method in the RenderedImageFactory interface;
- *Exploitation Framework Service:*
  - the addition of the notion of a per-use or per-DAG existence for Framework objects, rather than per-user;
  - the ability to see and use the IES interfaces and operations;
  - the changing of the name of the FrameworkMgr interface to FrameworkFactory;
- *Image Exploitation Service:*
  - the added responsibility (although already assumed) of publishing all image processing operations; and
  - the notions of *public, private,* and *custom-built* operations;
  - the responsibility of informing the service broker of all new operations registering with the IES::OperationFactory.
- *Others:*
  - the availability within the GIAS of a way to access non-image data (e.g. the ArrayAccessManager);
  - the modification of the UCO::Product structure to include a string to be used as a query key, and a string for local identification.

As further research is carried out, more changes may be required. We have endeavoured to retain as much of the current GIXS as possible rather than modifying it.

5. Conclusion

We have presented a federated GIXS model that has been adapted from the current GIXS and which will interoperate with the USIGS architecture. A general description of the model has been provided along with an explanation of the flow of events in such a federated system for a particular scenario. This research is still in progress and there are a number of areas still to be investigated in detail. Nevertheless, the main concepts are in place and we believe this is a consistent and appropriate model for the purpose of distributing image exploitation chains across a network of heterogeneous computers, each varying not only in platform but also in specific capabilities.

Future work on this project includes the following areas and tasks:
- the construction of a working prototype that demonstrates the appropriate functionality (e.g. chaining together of various operations, distribution of a single

DAG across multiple heterogeneous computers, failure recovery, DAG persistence, and private image exploitation services such as a RemoteView™ service);
- development and evaluation of algorithms for traversing and dividing DAGs for use by the Workflow Managers;
- the use of XML to make DAGs persistent and thus reusable;
- the investigation of issues associated with the manipulation of distributed DAGs in mid-execution such as failure recovery, synchronisation and consistency, parallelisation of processing, load balancing, and performance;
- the investigation of techniques for allowing XQMgrs to refuse work;
- the investigation of what information and metadata is to be used to register services with the service broker, and how the metadata framework should be implemented (initial work indicates XML may be a good option); and
- the investigation of what form references passed between the Buffer, WFMgrs, and Frameworks should take.

6. Addendum

A minimal implementation of the FGIXS has been developed using Java 1.2.2, Inprise VisiBroker 4.0, and Java Advanced Imaging 1.0.2. It has been successfully run on multiple heterogeneous hosts, including Solaris 2.7, Linux 6.2 Windows NT 4.0, and Windows 2000. It carries out the processing for an automatic target detection algorithm embodied within a DAG. The target detection operation was developed for synthetic aperture radar imagery and is written in C. It was linked into the Java implementation via the Java Native Interface. Performance tests have been carried out and results indicate that the architecture introduces little overhead to the processing of the DAG, and that most execution time is taken up with the transferring of data via the CORBA middleware implementation and Java Advanced Imaging library routines.

Furthermore, work on a second generation of the architecture investigating areas mentioned above in section 5 has begun. Particular effort has been placed in the development of the structure and semantics of the DAG given that the WFMgr and Framework components rely so heavily upon them. Other areas that are being investigated are tiling, caching, and tile streaming mechanisms, the incorporation of robustness and recovery features, and pull and push execution models in the same architecture. The new CORBA Component Model and Java 2 Enterprise Edition are being investigated also.

Appendix A: Abbreviations and Acronyms

This appendix lists the abbreviations and acronyms used in this document.

Acronym or Abbreviation	Definition
Buffer	Exploitation Buffer
CORBA	Common Object Request Broker Architecture
DAG	Directed Acyclic Graph, specifically in this document of image processing or exploitation operations
FW	Exploitation Framework
GIAS	Geospatial and Imagery Access Service
GIXS	Geospatial Information and Imagery Exploitation Service
IDL	Interface Description Language, as defined by the International Standards Organisation
IES	Image Exploitation Service
IMAD	Image Management and Dissemination (system)
IPL	Image Product Library
NIMA	National Imagery and Mapping Agency (of the United States of America)
RemoteView™	An image exploitation tool
RMI	Java Remote Method Invocation
Task	A DAG of image exploitation or processing operations
UCO, UCOS	USIGS Common Objects Specification, defines general-use data structures
UID::Product	USIGS Identification object, used to retrieve data from GIAS-based systems
USIGS	United States Imagery and Geospatial Information Service
WFMgr	Exploitation Workflow Manager
XDAG	Exploitation DAG, used in second generation of FGIXS architecture
XQMgr	Exploitation Query Manager

Appendix B: XD module IDL

```
#include <uco.idl>

//
// The XD Module contains the data structure definitions for the FGIXS
// directed acyclic graph, used for representing chains of image processing
// and exploitation operations.
//
// author: Derek Weber
// date:   1st August 2000
//

module XD {

  // Admin XNode types
  // exploitation operation
  const string ADMIN_XOP = "ADMIN_XOP";

  // for activating callbacks to the client
  const string ADMIN_CALLBACK = "ADMIN_CALLBACK";

  // for loading files
  const string ADMIN_FINELOAD = "ADMIN_FILELOAD";

  // for storing files
  const string ADMIN_FILESTORE = "ADMIN_FILESTORE";

  // Push Initiation Node - for simulating the push execution model
  const string ADMIN_PIN = "ADMIN_PIN";

  // Remote source
  const string ADMIN_RSOURCE = "ADMIN_RSOURCE";

  // Remote sink
  const string ADMIN_RSINK = "ADMIN_RSINK";

  // Split node for minimising network traffic
  const string ADMIN_SPLIT = "ADMIN_SPLIT";


  // image source
  struct Source {
    string  name;
    string  type;
    string  source_string; // source param or filename
  };

  // list of Sources
  typedef sequence <Source> SourceList;


  // sequence of 8-bit bytes
  typedef sequence <octet> ByteSeq;

  // byte stream
  struct ByteValue {
    long     length;
    ByteSeq  byte_seq;
};

// parameter
struct Param {
  string   name;
  string   type;
  ByteSeq  value;
};

// list of parameters
typedef sequence <Param> ParamList;


// hint structure
struct Hint {
  string     name;
  ByteValue  value;
};

// list of hints
typedef sequence <Hint> HintList;


// output definition
struct Output {
  string  name;
  string  type;
};

// sequence of outputs
typedef sequence <Output> OutputList;


// DAG node data structure, represents an exploitation operation
// or is an administration node
struct XNode {
  NodeID      id;   // for internal use with node management
  string      type; // Admin node or XOP
  string      name; // operation name
  SourceList  sources;
  ParamList   params;
  HintList    hints;
  OutputList  outputs;
};

// list of XNodes
typedef sequence <XNode> XNodeList;


// parameter mapping - result of one XNode becomes the input of another
struct ParamMapping {
  string  src_param_name;
  string  sink_param_name;
  string  type;
};

// list of parameter mappings
typedef sequence <ParamMapping> MappingList;


// edge structure - identifies the XNodes involved and the parameter
```

```
  // mappings
  struct XEdge {
    NodeID       src;
    NodeID       sink;
    MappingList  param_mappings;
  };

  // list of XEdges
  typedef sequence <XEdge> XEdgeList;


  // list of NodeIDs
  typedef sequence <NodeID> NodeIDList;

  // NodeID/NodeIDList pairing
  struct NodeIDNodeIDList {
    NodeID      node;
    NodeIDList  list;
  };

  // list of NodeID/NodeIDList pairings - effectively a table of NodeIDs
  // - used to determine the edges leading in and out of each XNode in the
  // XDAG
  typedef sequence <NodeIDNodeIDList> NodeIDNodeIDTable;


  // information about the exploitation task
  struct TaskInfo {
    string  user;
    string  group;
    string  job_id;
    string  priority;
    string  permissions;
  };


  // the XDAG structure
  struct XDAG {
    boolean            analysed; // true if XDAG already analysed
    XNodeList          nodes;
    XEdgeList          edges;
    NodeIDNodeIDTable  edges_in;
    NodeIDNodeIDTable  edges_out;
    HintList           hints;
    TaskInfo           task_info;
  };

} // end XD module
```

Acknowledgements

The authors would like to acknowledge the efforts of Ken Hawick and Paul Coddington from the Distributed & High Performance Computing group of the University of Adelaide and Don Uksi, Garry Hamlyn, and Kim Tang from the Image Analysis & Exploitation group of the Defence Science and Technology Organisation.

References

[1] *Geospatial and Imagery Exploitation Service (GIXS) Specification*, Version 1.0, National Imagery and Mapping Agency (NIMA), United States Imagery and Geospatial Information System (USIGS), document number S1010420-A, 22 June 1999.

[2] *USIGS Architecture Products Home Page*, National Imagery and Mapping Agency (NIMA), 9 February 2000. Available via the WWW as http://www.nima.mil/sandi/arch/.

[3] P.D. Coddington, K.A. Hawick, K.E. Kerry, J.A. Mathew, A.J. Silis, D.L. Webb, P.J. Whitbread, C.G. Irving, M.W. Grigg, R. Jana, & K. Tang, "Implementation of a Geospatial Imagery Digital Library using Java and CORBA", in *Proc. Technologies of Object-Oriented Languages and Systems Asia (TOOLS 27)*, IEEE, September 1998.

[4] Object Management Group, *The Object Management Group Home Page*, January 2000. Available via the WWW as http://www.omg.org/.

[5] Sun Microsystems, *Java™ Remote Method Invocation*, January 2000. Available via the WWW as http://www.javasoft.com/products/jdk/rmi/index.html.

[6] *Geospatial and Imagery Access Service (GIAS) Specification*, Version 3.3, National Imagery and Mapping Agency (NIMA), United States Imagery and Geospatial Information System (USIGS), document number N0101-E, 22 June 1999.

[7] OpenGIS Consortium, *Topic 15: Abstract Specification Image Exploitation Services*, Wayland, Massachussetts, USA, 1999. Available via the WWW as http://www.opengis.org/techno/specs.htm.

[8] OpenGIS Consortium, *Topic 0: Abstract Specification Overview*, Wayland, Massachusetts, USA, 1999. Available via the WWW as http://www.opengis.org/techno/specs.htm.

[9] Sun Microsystems, *The Java™ Advanced Imaging Home Page*, January 2000. Available via the WWW as http://www.javasoft.com/products/java-media/jai/.

[10] Sensor Systems Incorporated, *Welcome to RemoteView!*, January 2000. Available via the WWW as http://www.sensor.com/html3/remoteview.html.

[11] OpenGIS Consortium, *Topic 13: Abstract Specification Catalog Services*, Wayland, Massachussetts, USA, 1999. Available via the WWW as http://www.opengis.org/techno/specs.htm.

DISTRIBUTION LIST

A Federated GIXS Model

Derek Weber & Heath James

AUSTRALIA

DEFENCE ORGANISATION

Task Sponsor DAIO attention: Staff Officer Science

S&T Program

Chief Defence Scientist
FAS Science Policy } shared copy
AS Science Corporate Management
Director General Science Policy Development
Counsellor Defence Science, London (Doc Data Sheet)
Counsellor Defence Science, Washington (Doc Data Sheet)
Scientific Adviser to MRDC Thailand (Doc Data Sheet)
Scientific Adviser Policy and Command
Navy Scientific Adviser (Doc Data Sheet and distribution list only)

Scientific Adviser - Army (Doc Data Sheet and distribution list only)

Air Force Scientific Adviser
Director Trials

Aeronautical and Maritime Research Laboratory
Director

Electronics and Surveillance Research Laboratory
Director (Doc Data Sheet and distribution list only)
Chief of Surveillance Systems Division
Research Leader Imagery Systems
Head IAE Group
Task Manager JNT 99/016
Author(s):
Derek Weber
Heath James

DSTO Fernhill
Mark Grigg

DSTO Library and Archives
Library Fishermans Bend (Doc Data Sheet only)
Library Maribyrnong (Doc Data Sheet only)
Library Salisbury
Australian Archives
Library, MOD, Pyrmont (Doc Data Sheet only)
US Defense Technical Information Center, 2 copies
UK Defence Research Information Centre, 2 copies

Canada Defence Scientific Information Service, 1 copy
NZ Defence Information Centre, 1 copy
National Library of Australia, 1 copy

Capability Development Division
Director General Maritime Development (Doc Data Sheet only)
Director General Land Development (Doc Data Sheet only)
Director General C3I Development (Doc Data Sheet only)
Director General Aerospace Development (Doc Data Sheet only)

Army
ABCA Standardisation Officer, Puckapunyal, (4 copies)
SO (Science), DJFHQ(L), MILPO Enoggera, Queensland 4051 (Doc Data Sheet only)
NAPOC QWG Engineer NBCD c/- DENGRS-A, HQ Engineer Centre Liverpool Military Area, NSW 2174 (Doc Data Sheet only)

Intelligence Program
DGSTA Defence Intelligence Organisation
Manager, Information Centre, Defence Intelligence Organisation

Corporate Support Program
OIC TRS, Defence Regional Library, Canberra

UNIVERSITIES AND COLLEGES
Australian Defence Force Academy
Library
Head of Aerospace and Mechanical Engineering
Serials Section (M list), Deakin University Library, Geelong, 3217
Senior Librarian, Hargrave Library, Monash University (Doc Data Sheet only)
Librarian, Flinders University
University of Adelaide, Department of Computer Science
Ken Hawick
Paul Coddington

OTHER ORGANISATIONS

NASA (Canberra)
AGPS
State Library of South Australia
Parliamentary Library, South Australia
Sun Microsystems
Michael Bukva
Kevin Mayo
John Noonan
Roger Day

OUTSIDE AUSTRALIA

ABSTRACTING AND INFORMATION ORGANISATIONS
Library, Chemical Abstracts Reference Service

Engineering Societies Library, US
Materials Information, Cambridge Scientific Abstracts, US
Documents Librarian, The Center for Research Libraries, US

INFORMATION EXCHANGE AGREEMENT PARTNERS
Acquisitions Unit, Science Reference and Information Service, UK
Library - Exchange Desk, National Institute of Standards and Technology, US

Defence Evaluation Research Agency
Paul Hopkins

National Imagery and Mapping Agency
NIMA OGC and OMG Program Manager
David Lutz

SPARES (8 copies)

Total number of copies: 62

Page classification: UNCLASSIFIED

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA	1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)
2. TITLE A Federated Geospatial and Imagery Exploitation Service (GIXS) Model	3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION) Document (U) Title (U) Abstract (U)
4. AUTHOR(S) Derek Weber and Heath James	5. CORPORATE AUTHOR Electronics and Surveillance Research Laboratory PO Box 1500 Salisbury SA 5108 Australia

6a. DSTO NUMBER DSTO-TR-1013	6b. AR NUMBER AR-011-533	6c. TYPE OF REPORT Technical Report	7. DOCUMENT DATE August 2000

8. FILE NUMBER 9505/019/0039/01(U)	9. TASK NUMBER JNT 99/016	10. TASK SPONSOR DAIO	11. NO. OF PAGES 21	12. NO. OF REFERENCES 11

13. URL ON THE WORLD WIDE WEB http://www.dsto.defence.gov.au/corporate/reports/DSTO-TR-1013.pdf	14. RELEASE AUTHORITY Chief, Surveillance Systems Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, SALISBURY, SA 5108

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CASUAL ANNOUNCEMENT Yes

18. DEFTEST DESCRIPTORS

Digital image processing, Systems integration, Computer architecture

19. ABSTRACT

In order for the Geospatial and Imagery Exploitation Service (GIXS) architecture to take advantage of distributed processing of image exploitation tasks, it needs to be adapted to suit a federated environment. This document reports on work in progress by the Image Analysis and Exploitation Group in conjunction with the Distributed and High Performance Computing Group of The University of Adelaide to develop a federated GIXS architecture along with a proof-of-concept implementation.
A federated GIXS model is described, along with a use case scenario including an event-flow diagram. Also described are the changes necessary to adapt the current GIXS standard to our federated model. The report concludes with some future directions for our research.

Page classification: UNCLASSIFIED